Topic: Divide and Conquer

Theory: Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, $1< k \leq n$, yielding k sub problems. These sub problems must be solved and then a method must be found to combine sub solutions into a solution of the whole. If the sub problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied. Often the sub problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases the reapplication of the divide-and-conquer principle is naturally expressed by a recursive algorithm. Now smaller and smaller sub problems of the same kind are generated until eventually sub problems that are small enough to be solved without splitting are produced.

Control Abstraction:

```
Type DAndC(Problem P)
{
if small (P) return S(P);
else{
        divide P into smaller instances P1, P2, …. ,Pk, k ≥1;
        Apply  DAndC to each of these sub problems;
        Return combine(DAndC(P1), DAndC(P2),…., DAndC(Pk));
        }
```

}

Batch: A1 Roll No.: 1911013

Experiment No: 2

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of Binary search/Max-Min algorithm

Objective: To learn the divide and conquer strategy of solving the problems of different types

CO to be achieved:

Sr. No	Objective
CO 1	Analyze the asymptotic running time and space complexity of algorithms.
CO 2	Describe various algorithm design strategies to solve different problems and analyze Complexity.
CO 3	Develop string matching techniques
CO 4	Describe the classes P, NP, and NP-Complete

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Binary_search_algorithm**
4. **https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Binary_search_algorithm.html**
5. **http://video.franklin.edu/Franklin/Math/170/common/mod01/binarySearchAlg.html**
6. **http://xlinux.nist.gov/dads/HTML/binarySearch.html**
7. **https://www.cs.auckland.ac.nz/software/AlgAnim/searching.html**

Pre Lab/ Prior Concepts:

Data structures

Historical Profile:

Finding maximum and minimum or Binary search are few problems those are solved with the divide-and-conquer technique. This is one the simplest strategies which basically works on dividing the problem to the smallest possible level.

Binary Search is an extremely well-known instance of divide-and-conquer paradigm. Given an ordered array of n elements, the basic idea of binary search is that for a given element , "probe" the middle element of the array. Then continue in either the lower or upper segment of the array, depending on the outcome of the probe until the required (given) element is reached.

New Concepts to be learned:

Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.

Algorithm IterativeBinarySearch

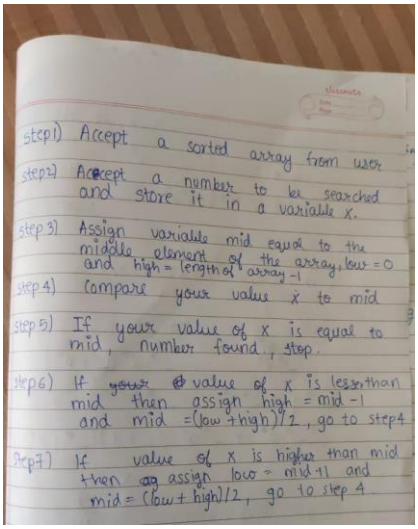
```
int binary_search(int A[ ], int key, int imin, int imax)
//The algorithm takes as parameters an array A[1.. n] , the search key and lower-higher index pair of the array.
 // Output- The algorithm returns index of the search key in the given array, if it's present.
{
  // continue searching while [imin, imax] is not empty
  WHILE (imax >= imin)
```

```
  {
          // calculate the midpoint for roughly equal partition
          int imid = midpoint(imin, imax);
          IF(A[imid] == key)
              // key found at index imid
              return imid;
            // determine which subarray to search
          ELSE If (A[imid] < key)
              // change min index to search upper subarray
              imin = imid + 1;
            ELSE
              // change max index to search lower subarray
              imax = imid - 1;
  }
 // key was not found
 RETURN KEY_NOT_FOUND;
}
```

The space complexity of Iterative Binary Search:

O(1)

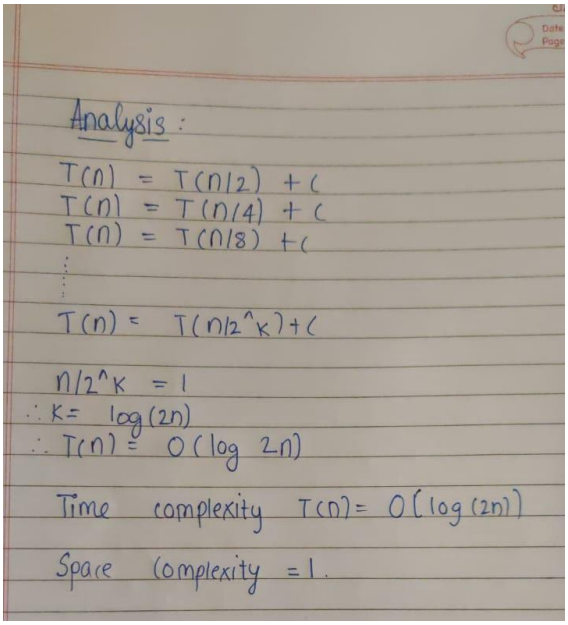
Algorithm RecursiveBinarySearch



The space complexity of Recursive Binary Search:

O(1)

The Time complexity of Binary Search:



Binary Search:

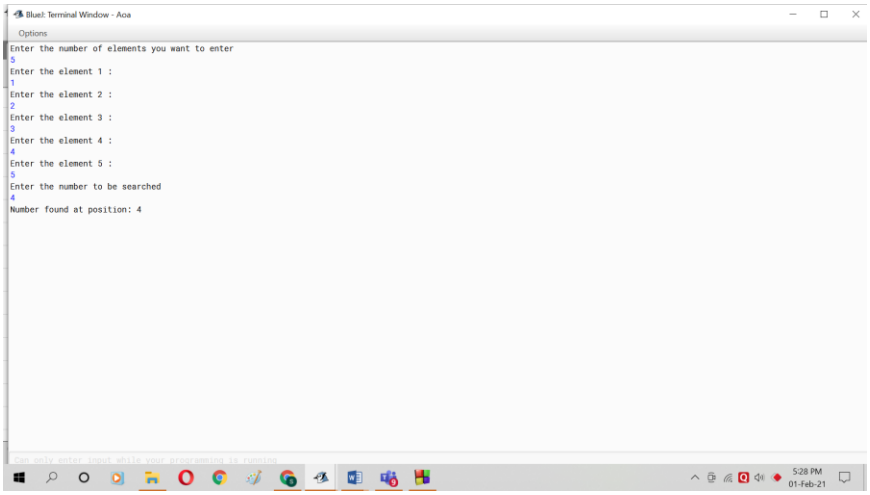
Using while loop:

Code:

```
import java.util.*;

public class binary_sort{

   public static void main(String Ars[]){

      Scanner sc=new Scanner(System.in);

      System.out.println("Enter the number of elements you want to enter");

      int n = sc.nextInt();

      int min=0;

      int max=n-1;

      int mid = (min+max)/2;

      int[] arr = new int[n];

      int chk=0;

      for(int i =0;i<n;i++)

      {

         System.out.println("Enter the element "+ (i+1)+" : ");

         arr[i]=sc.nextInt();

      }

      System.out.println("Enter the number to be searched");

      int x = sc.nextInt();

      while(min<=max && chk==0){
```

```
        if(x>arr[mid]){
           min=mid+1;
           mid = (min+max)/2;
        }
        else if(x<arr[mid]){
           max=mid-1;
           mid = (min+max)/2;
        }
        else{
           System.out.println("Number found at position: "+(mid+1));
           chk=1;
        }
      }
      if(chk==0)
        System.out.println("Number not found");
   }
}
```

Output:



Using Recursive Method:

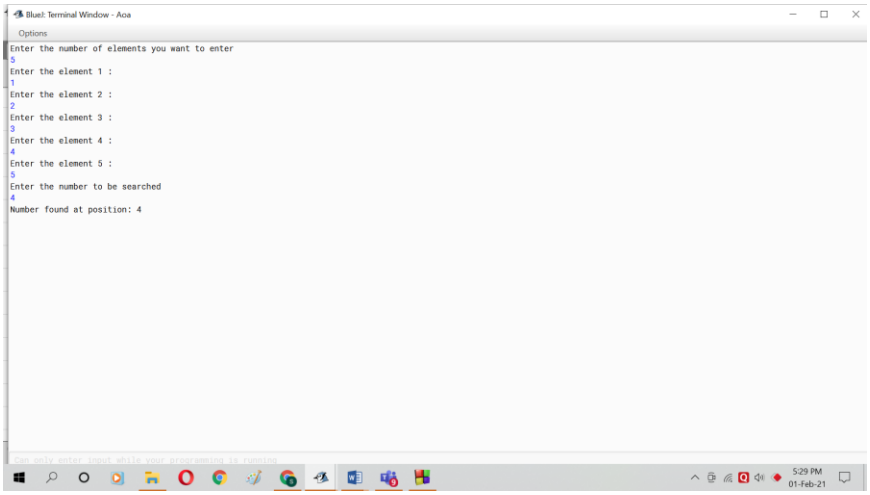
Code:

```
import java.util.*;

public class binary_sort_recursion{

   public static void main(String Ars[]){

      Scanner sc=new Scanner(System.in);

      System.out.println("Enter the number of elements you want to enter");

      int n = sc.nextInt();

      int min=0;

      int max=n-1;

      int mid = (min+max)/2;

      int[] arr = new int[n];

      int chk=0;
```

```
    for(int i =0;i<n;i++)
    {
       System.out.println("Enter the element "+ (i+1)+" : ");
       arr[i]=sc.nextInt();
    }
    System.out.println("Enter the number to be searched");
    int x = sc.nextInt();
    int ans=0;
    ans = search(min,max,x,arr);
    if (ans==0)
       System.out.println("Number not found");
}
public static int search(int min, int max, int x, int arr[]){
    int mid =(min+max)/2;
    if (min>=max){
       return 0;
    }
    else{
        if(x>arr[mid])
    {
       return search(mid+1, max, x, arr );
    }
```

```
    else if(x<arr[mid])
    {
       return search(min,mid-1 , x, arr );
    }
    else
    {
       System.out.println("Number found at position: "+(mid+1));
       return 1;
    }
    }


  }
}
```

Output:



Algorithm StraightMaxMin:

VOID StraightMaxMin (Type a[], int n, Type& max, Type& min)
// Set max to the maximum and min to the minimum of a[1:n].
{ max = min = a[1];

FOR (int i=2; i<=n; i++){

IF (a[i]>max) then max = a[i];

IF (a[i]<min) min = a[i];
}

}

Algorithm: Recursive Max-Min

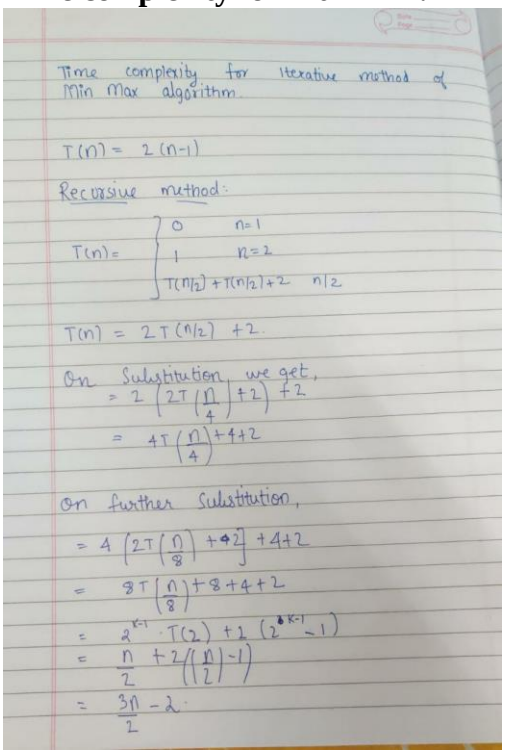
VOID MaxMin(int i, int j, Type& max, Type& min)
// A[1:n] is a global array. Parameters i and j are integers, 1 <= i <= j <= n.
//The effect is to set max and min to the largest and smallest values in a[i:j], respectively.
{
IF (i == j) max = min = a[i]; // Small(P)

```
    ELSE IF (i == j-1) { // Another case of Small(P)
          IF (a[i] < a[j])
                max = a[j]; min = a[i];
          ELSE { max = a[i]; min = a[j];
          }
       ELSE {      Type max1, min1;

// If P is not small   divide P into subproblems.  Find where to split the set.

           int mid=(i+j)/2;
          // Solve the  subproblems.

           MaxMin(i, mid, max, min);

           MaxMin(mid+1, j, max1, min1);

        // Combine the solutions.
          IF (max < max1) max = max1;
          IF (min > min1) min = min1;

   }
 }
```

The space complexity of Max-Min:

O(1)

Time complexity for Max-Min:



Minmax:

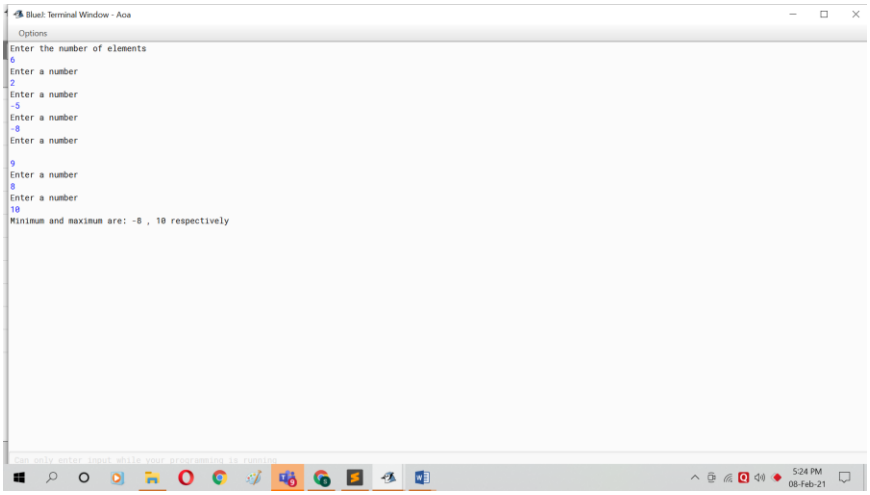
Using Iterative method:

Code:

```
import java.util.*;

public class MinMax{

   public static void main(String Args[]){

      Scanner sc=new Scanner(System.in);

      System.out.println("Enter the number of elements");

      int n = sc.nextInt();
```

```
    int i;
    int[] arr = new int[n];
    for(i=0;i<n;i++){
       System.out.println("Enter a number");
       arr[i]=sc.nextInt();
    }
    int min=arr[0];
    int max=arr[0];
    for(i=1;i<n;i++){
       if(min>arr[i])
          min=arr[i];
       else if(max<arr[i])
          max=arr[i];
    }
    System.out.println("Minimum and maximum are: "+min+" , "+max+" respectively");
  }
}
```

Output:



Using Recursive Method:

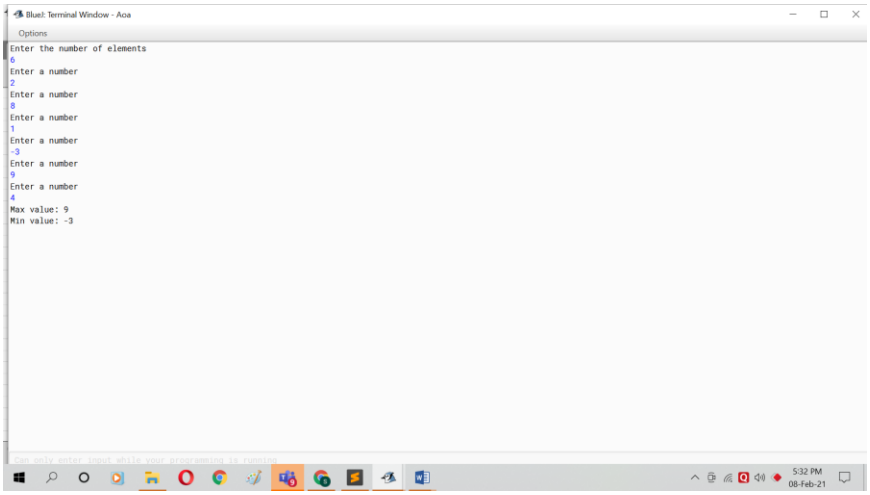
Code:

```
import java.util.*;

public class MinMax_recursive{

    static int max, min ;

    static int[] arr;

    public static void main(String Args[]){

       Scanner sc=new Scanner(System.in);

       System.out.println("Enter the number of elements");

       int n = sc.nextInt();

       int i;

       arr = new int[n];

       for(i=0;i<n;i++){
```

```
        System.out.println("Enter a number");
        arr[i]=sc.nextInt();
      }
      MaxMin(0,n-1);
      System.out.println("Max value: "+max+"\nMin value: "+min);
}
static void MaxMin(int i, int j)
      {
            int max1, min1, mid;
            if(i==j)
            {
                  max = min = arr[i];
            }
            else
            {
                  if(i == j-1){
                        if(arr[i] < arr[j]){
                              max = arr[j];
                              min = arr[i];
                        }
                        else{
                              max = arr[i];
```

```
                    min = arr[j];
                }
            }

            else{
                mid = (i+j)/2;
                MaxMin(i, mid);

                max1 = max;
                min1 = min;
                MaxMin(mid+1, j);
                if(max < max1) max = max1;//updating here
                if(min > min1) min = min1;
            }
        }
    }
}
```

Output:


BlueJ: Terminal Window - Aoa
Options
Enter the number of elements
6
Enter a number
2
Enter a number
8
Enter a number
1
Enter a number
-3
Enter a number
9
Enter a number
4
Max value: 9
Min value: -3
5:32 PM
08-Feb-21

CONCLUSION: